J. Bangladesh Electron. 15 (1-2); 89-101, 2015

Tree Based Incremental Sequential Pattern Mining

Md. Solaiman Mia

Department of Computer Science and Engineering, Hamdard University Bangladesh

Email: solaimanducse@yahoo.com

Abstract

Sequential pattern mining, which discovers the correlation relationships from the ordered list of events, is an important research field in data mining area. In this paper, I develop a Tree Based Incremental Sequential Pattern Mining algorithm which can generate sequential patterns from the Sequential Pattern Tree recursively. This algorithm builds a Sequential Pattern Tree for both frequent and non-frequent items. It requires only one scan of database to build the tree which can reduce the tree construction time considerably. The main advantage of this algorithm is to mine the complete set of sequential patterns from the Sequential Pattern Tree without generating any intermediate projected tree. It does not generate unnecessary candidate sequences and not require repeated scanning the original database. It also supports incremental mining. While the new sequences are updated, the already existing tree is revised for the updated sequences and then, mines the updated tree for the new frequent subsequences. I have compared the proposed approach with three state-of-the-art algorithms and the performance study shows that, Tree Based Sequential Pattern Mining algorithm is much faster than existing Apriori based GSP algorithm and also faster than existing PrefixSpan and FUSP-Tree algorithm. For incremental mining, the proposed approach outperforms both GSP and PrefixSpan.

Keyword: Sequential Pattern Mining, Data Mining, Sequence Database, Incremental Mining.

1. INTRODUCTION

Sequential pattern mining in large transactional databases plays an important role in data mining field. Sequential pattern mining is the process of finding the complete set of frequent occurring ordered events or subsequences from a set of sequences or sequence database. It is widely used in the analysis of customer purchase patterns or web access patterns, sequencing or time-related processes such as scientific experiments, natural disasters, and in DNA sequences, etc. Agrawal and Srikant first introduced sequential pattern mining in 1995 [1]. Past researches developed two major classes of sequential pattern mining methods. First class proposed several mining algorithms [1-2] based on Apriori property. Another class proposed algorithms like FreeSpan [3] and FUSP (Fast Updated Sequential Pattern) tree [4] based on pattern growth approach. FUSP tree which is proposed by Lin in 2008 [4], first generates a tree to store only frequent items and then, mine the original tree by reconstructing intermediate projected trees.

All of the above discussed sequential pattern mining algorithms [1-4] work in a one-time fashion: mine the entire database and obtain the complete set of frequent sequential patterns. However, in many applications, databases are updated incrementally. These algorithms [1-4] are not suitable for incremental mining. Because, the sequential patterns for the old database may become invalid on the updated database and it is not efficient to mine the updated databases from scratch by scanning the whole updated database (old + new).

In this paper, I have developed an efficient Tree Based Incremental Sequential Pattern Mining Algorithm for sequence database. At first, this algorithm generates a Sequential Pattern Tree

from the sequence database. It requires only one scan of database to build the tree which will reduce the tree construction time considerably. During mining, the Apriori-based sequential pattern mining algorithms [1-2] generate huge number of candidate patterns and pattern growth approaches [3-4] recursively generate a lot of projected databases which are both space and time consuming. But, the proposed algorithm will mine the complete set of sequential patterns from the original Sequential Pattern Tree without generating any intermediate projected tree. The proposed approach also supports incremental mining due to store non-frequent items. When new sequences come, it updates the original Sequential Pattern Tree by scanning only the new sequences, does not require scanning the whole updated database (old + new) and then, mine the updated new Sequential Pattern Tree from the beginning to retrieve the new frequent sequential patterns.

The paper is organized as follows: Section *2* contains the brief descriptions of some pattern mining algorithms. Section *3* describes the proposed Tree Based Incremental Sequential Pattern Mining approach. Section *4* concentrates on the experimental results of the findings and comparisons of the proposed approach with existing approaches using various datasets. Finally, Section *5* concludes this research work with an overview of the scopes for future studies.

2. BACKGROUND STUDY

I have studied a set of mining approaches to understand the effectiveness of pattern discovery in data mining field. The problem definition and some of these mining algorithms are described sequentially in this Section.

2.1 Problem Definition

Let, $I = \{i_1, i_2, ..., i_n\}$ be a set of items. A subset, X of items $(X \subseteq I)$ is called an *itemset*. A sequence is an ordered list of itemsets. A sequence $s = (s_1, s_2, \ldots, s_m)$ where s_i is an *itemset* $(s_i \subseteq I)$. s_i is also called an element which is denoted as $(x_1, x_2, \ldots, x_i)$ where x_j is an item. The number of items in a sequence is called the length of the sequence. A sequence with length l is called an l-sequence [3].

A database D is a set of tuples *(cid, tid, X),* where *cid* is the customer id, *tid* is the transaction id based on the transaction time, and X is an *itemset*. Each tuple in database, D is known as a transaction. All the transactions with the same *cid* create a sequence of the itemsets ordered by increasing *tid* [5].

The absolute support of a sequence s in a sequence database, D is the number of sequences in D that contain s. The relative support of a sequence s in a sequence database, D is the percentage of sequences in D that contain s. Given a minimum support threshold *min_sup*, a sequence s is called a frequent sequential pattern in D if support of $s \geq min\_sup$. Sequential pattern with length l is called an l-pattern [5].

2.1.1 Example of Sequence Database

Table 1 shows a set of tuples *(cid, tid, itemset)* for the transaction dataset. It is sorted by *cid* and then *tid.* Table 2 shows the corresponding sequences for each customer that is shown in Table 1.

Table 1: Dataset [5]

CID	*TID*	*Itemset*
1	1	*{a,b,d}*
1	3	*{b,c,d}*
1	6	*{b,c,d}*
2	2	*{b}*
2	4	*{a,b,c}*
3	5	*{a,b}*
3	7	*{b,c,d}*

Table 2: Sequence Database [5]

CID	*Sequence*
1	*({a,b,d} {b,c,d} {b,c,d})*
2	*({b} {a,b,c})*
3	*({a,b} {b,c,d})*

In Table 2, first sequence, s_1 = *({a, b, d} {b, c, d} {b, c, d})* has three elements or itemsets: *{a, b, d}, {b, c, d},* and *{b, c, d}* and *4* items: *{a, b, c, d}.* Items *b, c,* and *d* appear more than once in different elements. This sequence has *9* items. So, it is called *9*-sequence. Item *b* appears three times in this sequence, so it contributes 3 to the length of the sequence. However, the whole sequence *({a, b, d} {b, c, d} {b, c, d})* contributes only *1* to the support of *(b).* Suppose s_a = *({b} {b, c}).* s_a is a subsequence of s_1 and s_1 is a supersequence of s_a. Since both the second and third sequences also contain subsequence s_a, the support of s_a is *3*. So, s_a is frequent sequential pattern of length *3* (i.e., *3*-pattern).

2.2 FP-Growth Algorithm

FP-Growth algorithm [6] was proposed by Han, Pie & Yin, 2000, to mine the frequent patterns without candidate set generation. To mine the frequent patterns they proposed two novel algorithms, one is FP-tree construction algorithm and another is FP-Growth algorithm.

2.2.1 FP-Tree Construction Process

The construction process of the FP-tree was subdivided into two parts [6]. First, it scans the original database to find the frequent items and their frequency. Second, it scans the database from first transaction to last to complete the tree. The links between parent node and child node are singly directed. A header table was also kept to store the frequent items and links to the first occurrence of those items into the tree.

2.2.2 An Example of FP-Tree Structure

An example of FP-tree structure is shown in the Fig. 1.

Table 3: A Transaction Database [6]

TID	*Items*	*(Ordered) Frequent Items*
100	*f, a, c, d, g, i, m, p*	*f, c, a, m, p*
200	*a, b, c, f, l, m, o*	*f, c, a, b, m*
300	*b, f, h, j, o*	*f, b*
400	*b, c, k, s, p*	*c, b, p*
500	*a, f, c, e, l, p, m, n*	*f, c, a, m, p*

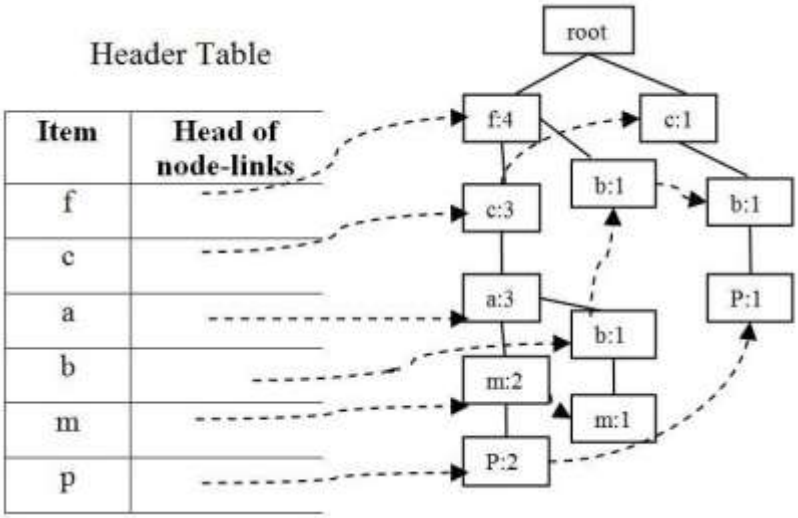

Fig. 1. FP-Tree of Table 3.

2.3 FUSP Tree

To efficiently mine the sequential patterns for incremental databases Lin et al. 2008 proposed the FUSP-tree [4] structure and its maintenance algorithm for both appended transactions and new customers.

2.3.1 Example of FUSP-Tree Structure

An example of FUSP-tree structure is shown in Fig. 2.

Table 4: Sequence Database

CID	*Sequences*
10	*({a} {abc} {ac} {d} {cf})*
20	*({ad} {c} {bc} {ae})*
30	*({ef} {ab} {df} {c} {b})*
40	*({e} {g} {af} {c} {b} {c})*

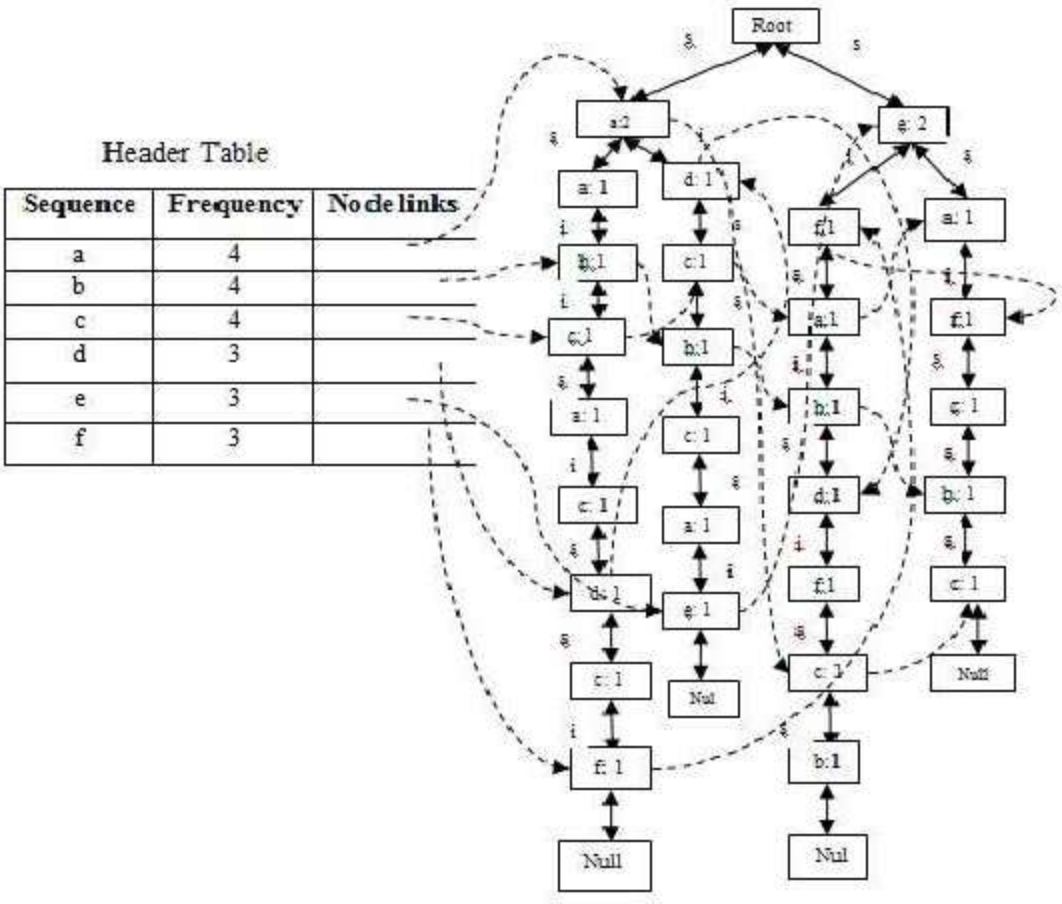

Fig. 2. FUSP-Tree of Table 4.

In Fig. 2, only the frequent length-*1* items are kept as like as FP-tree [6] and its construction process is also similar to FP-tree [6] i.e., the construction process is executed tuple by tuple from first customer sequence to last. But the differences from FP-tree are, the link between two nodes is symbolized by *'s'* or *'i'* as like IncSpan algorithm [7]. Here, symbol *'s'* indicates there exist sequence relation (*s-relation*) between two nodes and symbol *'i'* indicates that there exist the *itemset* relation (*i-relation*) between two nodes and the links are bidirectional which will help to update process easier.

3. PROPOSED TREE BASED INCREMENTAL SEQUENTIAL PATTERN MINING APPROACH

In this Section, I have described the proposed Tree Based Incremental Sequential Pattern Mining approach which is a sequential pattern mining algorithm for sequence database. At first, the proposed approach constructs a Sequential Pattern Tree for both frequent and non-frequent items from the sequence database. Then, it generates the complete set of frequent subsequences from the Sequential Pattern Tree without generating any intermediate projected tree. This algorithm also supports incremental mining. While new sequences are appended to the old database, the already existing tree is revised for the updated sequences. Then, it initiates the sequential pattern mining from the beginning. The algorithm for Tree Based Sequential Pattern mining is given in Algorithm 1.

Algorithm 1: (Tree Based Sequential Pattern Mining).

Input: Sequence Database and Minimum Support threshold (*min_sup*).

Output: The complete set of sequential patterns.

Method:

1. Scan the sequence database once and construct a Sequential Pattern Tree using Algorithm 2.
2. Recursively mine the Sequential Pattern Tree to find the frequent sequential patterns using Algorithm 3.

3.1 Construction of Sequential Pattern Tree

In this paper, a Sequential Pattern Tree data structure along with header table is used to store the sequence database. This tree stores both frequent and non-frequent items. So, it requires only one scan of database to build the tree which reduces the tree construction time considerably. Transaction ID in a node is used to indicate if there exists sequence relation (*s-relation*) or *itemset* relation (*i-relation*) between two nodes. The root of the tree is a special virtual node with a label as Root, count *0*, and transaction ID *0*.

The Sequential Pattern Tree is constructed as follows: Scans the sequence database and insert each item in the events of a sequence into the tree. The insertion of sequences is started from the root node of the tree. For each item *e* in the events, increment the count of child node with label *e* by *1* if there exists one child node with same label; otherwise, create a child node labeled by *e* and set the count to *1*. The algorithm for constructing a Sequential Pattern Tree from sequence database is given in Algorithm 3.

Algorithm 2: (Construction of Sequential Pattern Tree from Sequence Database).

Input: Sequence Database.

Output: Sequential Pattern Tree, *T* along with Header Table.

Method:

1. Scan the sequence database.
2. Create the root node of a tree *T* and label it as "Root", set count to *0* and transaction ID to *0*. Initially *current_node = root*.
3. **for** each sequence S_i till the end of database
    - **for** each event e_j in S_i
        - **for** each item I in the e_j
            - **if** current_node has a child node *c* and *c.label = I* and *c.transaction ID = j*, **then**, *c.count* += *1* and *current_node = c*.
            - otherwise,
                - Create a New node label with *I*
                - New *node.count = 1*
                - New *node.transaction ID = j*
                - Store New node in the current_node's successor link.
                - Set Current_node = New node
            - **end if**
            - **if** the branch is new for the item *I* then increment the count of the corresponding item *I* in the header table if item *I* already

exist; otherwise, add item *I* in the header table and set count to *1*.

- **end for**

- **end for**

4. **end for**

3.1.1 Example of Construction of Sequential Pattern Tree

In this Subsection, I will try to describe the algorithm for construction of Sequential Pattern Tree by using an example. As input, the proposed algorithm just takes a sequence database. In this example, I have used a sequence database which is shown in Table 5.

Table 5: Original Sequence Database

Sequence ID	***Sequences***
10	*a(abc)(ac)*
20	*(ad)c(ae)*
30	*d(cf)(bc)*
40	*(ab)(ad)*

The Sequential Pattern Tree for this sequence database is shown in Fig. 3, which is constructed as follows. Scan the database and find the first sequence *a(abc)(ac).* Insert this sequence into the initial tree with only one root node. It creates a new node *(a: 1: 1)* (i.e., labeled as *a*, with count set to *1* and transaction ID to *1*) as the child of the root node, and then derives the *a*-branch *"(a: 1: 1) → (a: 1: 2) → (b: 1: 2) → (c: 1: 2) → (a: 1: 3) → (c: 1: 3)",* in which arrows point from parent nodes to children nodes. Now, insert the second sequence *(ad)c(ae).* It starts from the root. Since the root node has a child labeled with "*a*" and transaction ID of this node is also *1*, then, *a*'s count is increased by *1*, i.e., *(a: 2: 1)* now. But, next item, *d* in first event of second sequence does not match with the existing child node of node *(a: 2: 1).* So, create a new child node *(d: 1: 1)* of node *(a: 2: 1).* This process continues until there is no sequence in the sequence database.

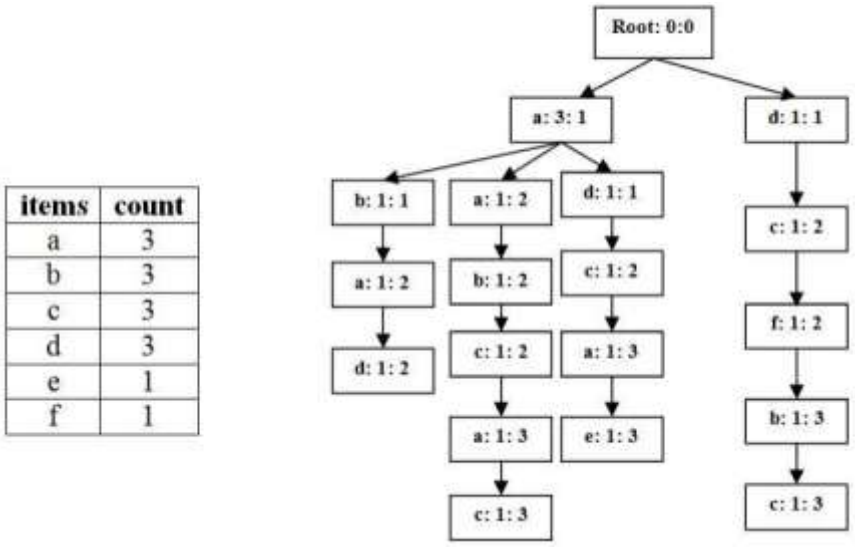

Fig. 3. Sequential Pattern Tree with Header Table.

3.2 Mining Sequential Patterns from Sequential Pattern Tree

In this Subsection, I have developed an efficient recursive algorithm to enumerate frequent sequential patterns from the Sequential Pattern Tree. The proposed algorithm uses the original Sequential Pattern Tree for the entire mining and does not rebuild intermediate trees for projection databases like FUSP tree [4] from the original tree during mining. The algorithm for mining frequent patterns from the Sequential Pattern Tree is described in Algorithm 3.

This algorithm starts from the header table. For each frequent item I in the header table, it always try to find the first-occurrence node with labeled I from each branch of the original tree and store these nodes in the *rootset*. The first-occurrence nodes of a symbol are found using depth-first-search of the tree. The algorithm of finding the first-occurrence node is given in Algorithm 4.

Algorithm 3: (Mining Sequential Patterns from Sequential Pattern Tree).

Input: Sequential Pattern Tree with Header Table and Minimum Support, *min_sup*.

Output: The Complete Set of Sequential Patterns.

Global Variable: *Rootset_s* to store *s-relation* nodes, *Rootset_i* to store *i-relation* nodes, Track to store each root node.

Other Variable: *F* to store frequent pattern.

Initial: *Rootset_s* stores root of the original tree. *F* set as null. At first, call the SP-tree Mine() of Algorithm 4 by passing *Rootset_s* and *F* as null.

Method:

- **for** each frequent item *I* in the header table
- *Rootset_s* = new *Rootset()*
- *Rootset_i* = new *Rootset()*
- **for** each root node *R* of the *Rootset*
    - *Track = R*
    - **for** each child node *N* of *R*
        - First-occurrence-node(*I, N, 0,0*) [Describe in Algorithm 4]
    - **end for**
- **end for**
- **if** (the sum of the count of root nodes in the *Rootset_s* $\geq$ *min_sup*), **then**
    - *F' = F* $\cup$ *(I)*
    - Call SP-tree Mine (*Rootset_s, F'*)
- **end if**
- **if** (the sum of the count of root nodes in the *Rootset_i* $\geq$ *min_sup*), **then**
    - *F' = (F* $\cup$ *I)*
    - Call SP-tree Mine (*Rootset_i, F'*)
- **end if**
- **end for**

Algorithm 4: (Finding First Occurrence Node that Labeled as *I* from Sequential Pattern Tree).

Input: Frequent Item, *I* and child node *N* of root node *R* from *Rootset*, *Mark_s* variable use to find only one *s-relation* node labeled as *I* from a branch and *Mark_i* variable use to find only

one *i-relation* node labeled as *I* from a branch.

Output: The First Occurrence nodes those Labeled as *I*.

Global Variable: Mark variable use to keep track if the parent node label of a node equal to the root node label. Initially, Mark set as *0*.

Method:

- **if** (*N. label = Track. label*)
  - set Mark as *N.transaction ID*
- **end if**
- **if** (*N.label = I*), **then**
  - **if** (*N. transaction ID = Track. transaction ID && Mark_i = 0*)
    - Append *N* to *Rootset_i*
    - *Mark_i* set as *1*
  - **else if** (*N.transaction ID != Track.transaction ID && Mark = N.transacttion ID && Mark_i = 0*)
    - Append *N* to *Rootset_i*
    - *Mark_i* set as *1*
  - **else if** (*N.transaction ID != Track.transaction ID && Mark != N.transacttion ID && Mark_s = 0*)
    - Append *N* to *Rootset_s*
    - *Mark_s* set as *1*
  - **end if**
- **end if**
- **for** each child node, *n* of node *N*
  - First-occurrence-node (*I, n, Mark_s, Mark_i*)
- **end for**

3.3 Incremental Sequential Pattern Mining

In this Subsection, I describe an incremental sequential pattern mining process. When new sequences come, each new sequence is inserted as branch into the original Sequential Pattern Tree and the items in the header table are also updated using the Algorithm 2 presented in Subsection 3.1. When sequence database is updated with new sequences, it is not required to scan the entire sequence database, only scan the new sequences to update the existing Sequential Pattern Tree and header table. Then, initiate the sequential pattern mining process from the beginning using the Algorithm 3 and Algorithm 4 that present in Subsection 3.2.

3.3.1 Example of Incremental Sequential Pattern Mining

In this Subsection, I describe the incremental sequential pattern mining with a suitable example. Table 6 shows incremental sequence database that is appended with the original sequence database shown in Table 5.

Table 6: Incremental Sequence Database

Sequence ID	***Sequences***
50	*a(ad)c(ae)*
60	*f(bc)(ae)*

Scan only the incremental sequence database once and insert the sequence *(ad)c(ae)* as existing branch into the tree shown in Fig. 3. Only count of each node of this branch increment. The next sequence *f(bc)(ae)* is inserted into the original tree as a new branch using Algorithm 3. Header table also update accordingly. The complete new updated Sequential Pattern Tree and updated header table is shown in Fig. 4.

Suppose the *min_sup* threshold for the updated sequence database is *50%* or *3 (6×50% = 3).* This time, item *e: 3* become frequent. Only item *f: 2* is non frequent. Initiate the sequential pattern mining process from the beginning of the Updated Sequential Pattern Tree shown in Fig. 4 using the Algorithm 3 and Algorithm 4 that present in Subsection 3.2 and find the new updated frequent sequential patterns are *{(a): 5, (a)(a): 4, (a)(c): 3, (a)(c)(a): 3, (a, d): 3, (a, e): 3, (b): 4, (b)(a): 3, (b, c): 3, (c): 5, (c)(a): 4, (c)(a, e): 3, (c)(e): 3, (d): 4, (d)(c): 3, (e): 3}*.

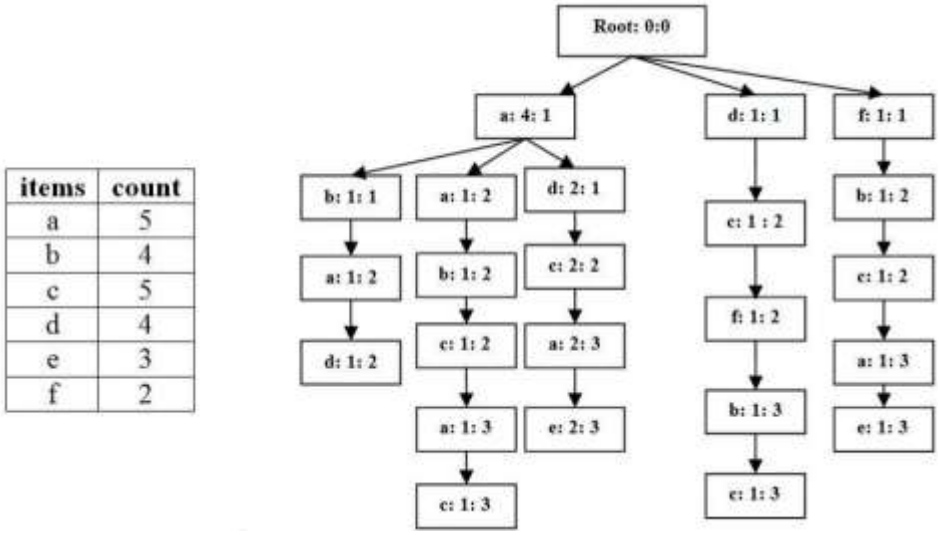

Fig. 4. Updated Sequential Pattern Tree along with Updated Header Table.

4. COMPARATIVE STUDY

In this Section, I represent a performance comparison of the proposed Tree Based Incremental Sequential Pattern Mining approach with GSP [1], PrefixSpan [6] and FUSP-Tree [4] on both synthetic and real-life datasets.

I have used two datasets, BMS-WebView-1 [8] and T10I4D100K [8] for evaluation of experimental results. The Properties of these datasets are shown below by Table 7.

Table 7: Properties of Experimental Datasets

Dataset	Distinct Items	No. of Sequences	Max. Size	Avg. Size	Type
BMS-WebView-1	497	59602	267	2.5	Real
T10I4D100K	870	100000	29	10.1	Synthetic

4.1 Experimental Results

In this Subsection, I have showed the experimental results for interactive mining using different minimum support threshold values and also for incremental mining for above two datasets.

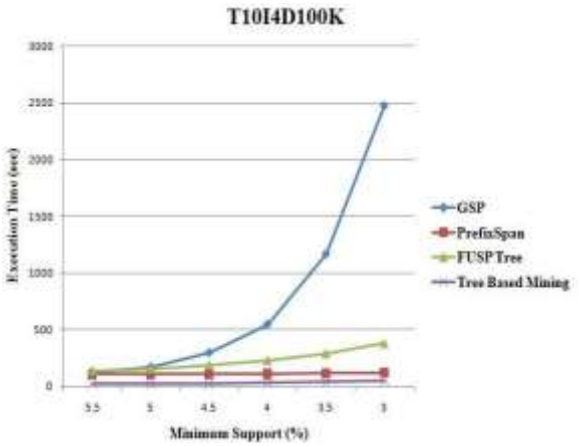

Fig. 5. Comparisons between Execution Time and Minimum Support for T10I4D100K.

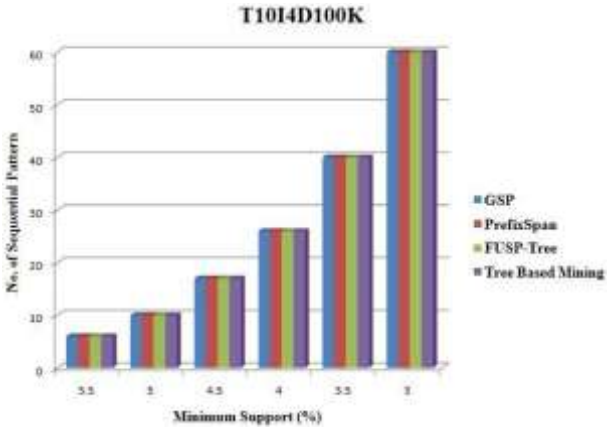

Fig. 6. Comparisons between No. of Sequential Patterns and Minimum Support for T10I4D100K.

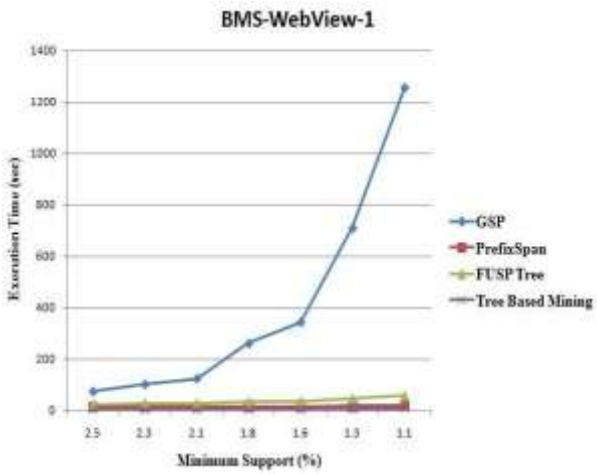

Fig. 7. Comparisons between Execution Time and Minimum Support for BMS-WebView-1.

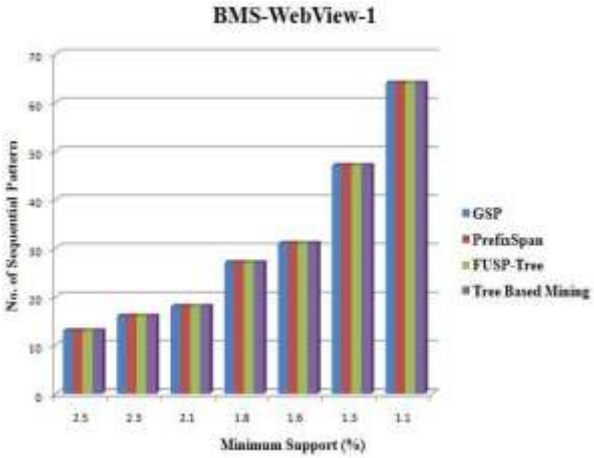

Fig. 8. Comparisons between No. of Sequential Patterns and Minimum Support for BMS-WebView-1.

The experimental results in Fig.5 and Fig. 7 are depicted to show the execution time of the four algorithms at different support thresholds. It can be observed from these Figures that, the execution times maintain the order "Tree based approach < PrefixSpan < FUSP-Tree < GSP" when T10I4D100K and BMS-WebView-1 datasets are used respectively. Thus, it can conclude that, the proposed Tree based sequential pattern mining approach performs much better than apriori based GSP algorithm and also outperforms PrefixSpan and FUSP-Tree which are pattern growth approaches. This is also to be mentioned that, the proposed Tree based approach generates same number of sequential patterns for different minimum support thresholds as generated by GSP, PrefixSpan, and FUSP-Tree algorithms shown in Fig. 6 and Fig. 8; which prove the soundness and completeness of the proposed algorithm.

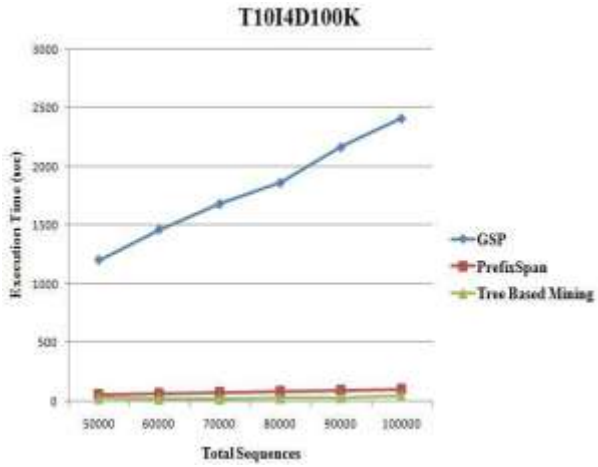

Fig. 9. Comparisons between Execution Time and Various Incremental Database Size for T10I4D1000K.

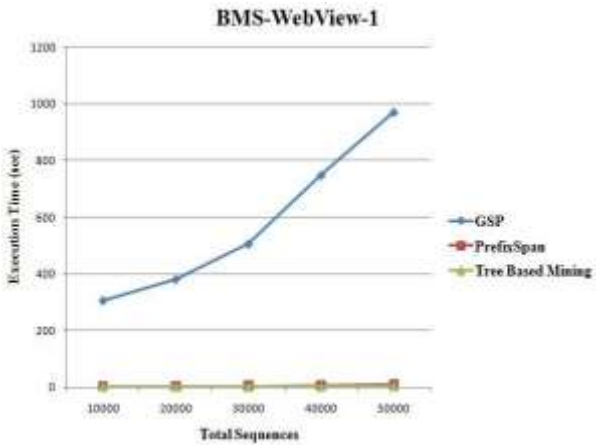

Fig. 10. Comparisons between Execution Time and Various Incremental Database Size for BMS-WebView-1.

The experimental results in Fig. 9 and Fig. 10 are depicted to show the execution time of the three algorithms at various incremental database sizes. It can be observed from these figures that, the execution time of the proposed Tree Based Incremental approach much better than GSP and PrefixSpan for both synthetic and real-datasets. Thus, it can conclude that, the proposed approach gives better performance than other two existing methods for incremental mining. Because, during incremental mining, the proposed approach update the old Sequential Pattern Tree by scanning only the new sequences once.

5. CONCLUSION

The proposed method can be used to find the sequential patterns from the sequence database. Previously many methods were proposed to find the sequential patterns. I have proposed an efficient sequential pattern mining algorithm which can generates the complete set of sequential patterns from a Sequential Pattern Tree without generating any candidate set and any intermediate projected tree that reduces the both space and time complexity. This approach first generates a Sequential Pattern Tree from the sequence database which stores both frequent and non-frequent items. So, it requires only one scan of sequence database to create tree along with header table which also reduces the tree construction time considerably. Although, GSP and PrefixSpan efficiently find the entire frequent sequential patterns, I have also proposed an incremental mining algorithm which updates the original Sequential Pattern Tree for the

incremental database by scanning only the incremental database once and then, mine the updated Sequential Pattern Tree to find the new frequent sequential patterns from the beginning. The proposed tree based approach works well than GSP and PrefixSpan for dynamically updated database. The proposed approach can also find out all the sequential patterns like GSP, PrefixSpan and FUSP-Tree. And it can also say that, the proposed method works better, than apriori based approach GSP and as well as than pattern growth approaches PrefixSpan and FUSP-Tree.

The proposed Sequential Pattern Tree stores both frequent and non-frequent items. So, it requires little more memory. It can be thought as a limitation but for storing both frequent and non-frequent items, it can achieve better performance during incremental mining and it requires only one scan of database to create tree and header table which reduces the mining time.

In the future, I will attempt to find other maintenance algorithm like deletion or modification of records for sequential pattern mining and I will also concentrate on other data mining areas.

REFERENCES

[1] R. Agrawal, and R. Srikant, "Mining sequential patterns", Proc. of the 11th International Conf. on Data Engineering, 1995, pp. 3-14, Taipei.

[2] M. J. Zaki, "Spade: An efficient algorithm for mining frequent sequences", Machine Learning, January 2001, vol. 42, no. 1, pp. 31-60.

[3] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu, "Freespan: frequent pattern-projected sequential pattern mining", Proc. of the 6th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining, 2000, pp. 355-359, USA.

[4] C. W. Lin, T. P. Hong, W. H. Lu, and W. Y. Lin, "An incremental FUSP-Tree maintenance algorithm", 8th International Conf. on Intelligent Systems Design and Applications , November 2008, vol. 1, pp. 445-449, Kaohsiung.

[5] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential PAttern Mining using a bitmap representation", Proc. of the 8th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining, 2002, pp. 429-435, USA.

[6] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation", Proc. of the 2000 ACM SIGMOD International Conf. on Management of Data, 2000, pp. 1-12, USA.

[7] H. Cheng, X. Yan, and J. Han, "Incspan: incremental mining of sequential patterns in large database", Proc. of the 10th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining, 2004, .pp. 527-532, USA.

[8] Z. Zheng, R. Kohavi, and L. Mason, "Real world performance of association rule algorithms", Proc. of the 7th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining, 2001, .pp. 401-406, USA.